Cardinality of Equivalence Relations

Lukas Bulwahn

March 17, 2025

Abstract

This entry provides formulae for counting the number of equivalence relations and partial equivalence relations over a finite carrier set with given cardinality.

To count the number of equivalence relations, we provide bijections between equivalence relations and set partitions [4], and then transfer the main results of the two AFP entries, Cardinality of Set Partitions [1] and Spivey's Generalized Recurrence for Bell Numbers [2], to theorems on equivalence relations. To count the number of partial equivalence relations, we observe that counting partial equivalence relations over a set A is equivalent to counting all equivalence relations over all subsets of the set A. From this observation and the results on equivalence relations, we show that the cardinality of partial equivalence relations over a finite set of cardinality n is equal to the $n+1$-th Bell number [3].

Contents

1 Cardinality of Equivalence Relations

```
theory Card-Equiv-Relations
imports
  Card-Partitions.Card-Partitions
  Bell-Numbers-Spivey.Bell-Numbers
begin
```

1.1 Bijection between Equivalence Relations and Set Partitions

1.1.1 Possibly Interesting Theorem for *HOL.Equiv-Relations*

This theorem was historically useful in this theory, but is now after some proof refactoring not needed here anymore. Possibly it is an interesting fact about equivalence relations, though.

```
lemma equiv-quotient-eq-quotient-on-UNIV:
  assumes equiv A R
  shows A // R = (UNIV // R) − {{}}
proof
  show UNIV // R − {{}} ⊆ A // R
  proof
    fix X
    assume X ∈ UNIV // R − {{}}
    from this obtain x where X = R `` {x} and X ≠ {}
      by (auto elim!: quotientE)
    from this have x ∈ A
      using ‹equiv A R› equiv-class-eq-iff by fastforce
    from this ‹X = R `` {x}› show X ∈ A // R
      by (auto intro!: quotientI)
  qed
next
  show A // R ⊆ UNIV // R − {{}}
  proof
    fix X
    assume X ∈ A // R
    from this have X ≠ {}
      using ‹equiv A R› in-quotient-imp-non-empty by auto
    moreover from ‹X ∈ A // R› have X ∈ UNIV // R
      by (metis UNIV-I assms proj-Eps proj-preserves)
    ultimately show X ∈ UNIV // R − {{}} by simp
  qed
qed
```

1.1.2 Dedicated Facts for Bijection Proof

```
lemma equiv-relation-of-partition-of:
  assumes equiv A R
  shows {(x, y). ∃X∈A // R. x ∈ X ∧ y ∈ X} = R
```

```
proof
  show {(x, y). ∃X∈A // R. x ∈ X ∧ y ∈ X} ⊆ R
  proof
    fix xy
    assume xy ∈ {(x, y). ∃X∈A // R. x ∈ X ∧ y ∈ X}
    from this obtain x y and X where xy = (x, y)
      and X ∈ A // R and x ∈ X y ∈ X
      by auto
    from ‹X ∈ A // R› obtain z where X = R `` {z}
      by (auto elim: quotientE)
    show xy ∈ R
      using ‹xy = (x, y)› ‹X = R `` {z}› ‹x ∈ X› ‹y ∈ X› ‹equiv A R›
      by (simp add: equiv-class-eq-iff)
  qed
next
  show R ⊆ {(x, y). ∃X∈A // R. x ∈ X ∧ y ∈ X}
  proof
    fix xy
    assume xy ∈ R
    obtain x y where xy = (x, y) by fastforce
    from ‹xy ∈ R› have (x, y) ∈ R
      using ‹xy = (x, y)› by simp
    have R `` {x} ∈ A // R
      using ‹equiv A R› ‹(x, y) ∈ R›
      by (simp add: equiv-class-eq-iff quotientI)
    moreover have x ∈ R `` {x}
      using ‹(x, y) ∈ R› ‹equiv A R›
      by (meson equiv-class-eq-iff equiv-class-self)
    moreover have y ∈ R `` {x}
      using ‹(x, y) ∈ R› ‹equiv A R› by simp
    ultimately have (x, y) ∈ {(x, y). ∃X∈A // R. x ∈ X ∧ y ∈ X}
      by auto
    from this show xy ∈ {(x, y). ∃X∈A // R. x ∈ X ∧ y ∈ X}
      using ‹xy = (x, y)› by simp
  qed
qed
```

1.1.3 Bijection Proof

```
lemma bij-betw-partition-of:
  bij-betw (λR. A // R) {R. equiv A R} {P. partition-on A P}
proof (rule bij-betw-byWitness[where f'=λP. {(x, y). ∃X∈P. x ∈ X ∧ y ∈ X}])
  show ∀R∈{R. equiv A R}. {(x, y). ∃X∈A // R. x ∈ X ∧ y ∈ X} = R
    by (simp add: equiv-relation-of-partition-of)
  show ∀P∈{P. partition-on A P}. A // {(x, y). ∃X∈P. x ∈ X ∧ y ∈ X} = P
    by (simp add: partition-on-eq-quotient)
  show (λR. A // R) ` {R. equiv A R} ⊆ {P. partition-on A P}
    using partition-on-quotient by auto
  show (λP. {(x, y). ∃X∈P. x ∈ X ∧ y ∈ X}) ` {P. partition-on A P} ⊆ {R.
```

equiv A R}
  using *equiv-partition-on* **by** *auto*
qed

lemma *bij-betw-partition-of-equiv-with-k-classes*:
 bij-betw (λ*R*. *A* // *R*) {*R*. *equiv A R* ∧ *card* (*A* // *R*) = *k*} {*P*. *partition-on A P* ∧ *card P* = *k*}
proof (*rule bij-betw-byWitness*[**where** *f'*=λ*P*. {(*x*, *y*). ∃ *X*∈*P*. *x* ∈ *X* ∧ *y* ∈ *X*}])
 show ∀ *R*∈{*R*. *equiv A R* ∧ *card* (*A* // *R*) = *k*}. {(*x*, *y*). ∃ *X*∈*A* // *R*. *x* ∈ *X* ∧ *y* ∈ *X*} = *R*
  by (*auto simp add*: *equiv-relation-of-partition-of*)
 show ∀ *P*∈{*P*. *partition-on A P* ∧ *card P* = *k*}. *A* // {(*x*, *y*). ∃ *X*∈*P*. *x* ∈ *X* ∧ *y* ∈ *X*} = *P*
  by (*auto simp add*: *partition-on-eq-quotient*)
 show (λ*R*. *A* // *R*) ' {*R*. *equiv A R* ∧ *card* (*A* // *R*) = *k*} ⊆ {*P*. *partition-on A P* ∧ *card P* = *k*}
  using *partition-on-quotient* **by** *auto*
 show (λ*P*. {(*x*, *y*). ∃ *X*∈*P*. *x* ∈ *X* ∧ *y* ∈ *X*}) ' {*P*. *partition-on A P* ∧ *card P* = *k*} ⊆ {*R*. *equiv A R* ∧ *card* (*A* // *R*) = *k*}
  using *equiv-partition-on* **by** (*auto simp add*: *partition-on-eq-quotient*)
qed

1.2 Finiteness of Equivalence Relations

lemma *finite-equiv*:
 assumes *finite A*
 shows *finite* {*R*. *equiv A R*}
proof −
 have *bij-betw* (λ*R*. *A* // *R*) {*R*. *equiv A R*} {*P*. *partition-on A P*}
  by (*rule bij-betw-partition-of*)
 from *this* **show** *finite* {*R*. *equiv A R*}
  using ‹*finite A*› *finitely-many-partition-on* **by** (*simp add*: *bij-betw-finite*)
qed

1.3 Cardinality of Equivalence Relations

theorem *card-equiv-rel-eq-card-partitions*:
 card {*R*. *equiv A R*} = *card* {*P*. *partition-on A P*}
proof −
 have *bij-betw* (λ*R*. *A* // *R*) {*R*. *equiv A R*} {*P*. *partition-on A P*}
  by (*rule bij-betw-partition-of*)
 from *this* **show** *card* {*R*. *equiv A R*} = *card* {*P*. *partition-on A P*}
  by (*rule bij-betw-same-card*)
qed

corollary *card-equiv-rel-eq-Bell*:
 assumes *finite A*
 shows *card* {*R*. *equiv A R*} = *Bell* (*card A*)
using *assms Bell-altdef card-equiv-rel-eq-card-partitions* **by** *force*

corollary *card-equiv-rel-eq-sum-Stirling*:
assumes *finite A*
shows *card* {*R. equiv A R*} = *sum* (*Stirling* (*card A*)) {..*card A*}
using *assms card-equiv-rel-eq-Bell Bell-Stirling-eq* **by** *simp*

theorem *card-equiv-k-classes-eq-card-partitions-k-parts*:
card {*R. equiv A R* ∧ *card* (*A* // *R*) = *k*} = *card* {*P. partition-on A P* ∧ *card P* = *k*}
proof −
have *bij-betw* (λ*R. A* // *R*) {*R. equiv A R* ∧ *card* (*A* // *R*) = *k*} {*P. partition-on A P* ∧ *card P* = *k*}
by (*rule bij-betw-partition-of-equiv-with-k-classes*)
from *this* **show** *card* {*R. equiv A R* ∧ *card* (*A* // *R*) = *k*} = *card* {*P. partition-on A P* ∧ *card P* = *k*}
by (*rule bij-betw-same-card*)
qed

corollary
assumes *finite A*
shows *card* {*R. equiv A R* ∧ *card* (*A* // *R*) = *k*} = *Stirling* (*card A*) *k*
using *card-equiv-k-classes-eq-card-partitions-k-parts*
card-partition-on[*OF* ‹*finite A*›] **by** *metis*

end

2 Cardinality of Partial Equivalence Relations

theory *Card-Partial-Equiv-Relations*
imports
Card-Equiv-Relations
begin

2.1 Definition of Partial Equivalence Relation

definition *partial-equiv* :: *'a set* ⇒ (*'a* × *'a*) *set* ⇒ *bool*
where
partial-equiv A R = (*R* ⊆ *A* × *A* ∧ *sym R* ∧ *trans R*)

lemma *partial-equivI*:
assumes *R* ⊆ *A* × *A sym R trans R*
shows *partial-equiv A R*
using *assms* **unfolding** *partial-equiv-def* **by** *auto*

lemma *partial-equiv-iff*:
shows *partial-equiv A R* ⟷ (∃ *A'* ⊆ *A*. *equiv A' R*)
proof
assume *partial-equiv A R*
from ‹*partial-equiv A R*› **have** *R* ‘‘ *A* ⊆ *A*
unfolding *partial-equiv-def* **by** *blast*

```
  moreover have equiv (R `` A) R
  proof (rule equivI)
    from ‹partial-equiv A R› show sym R
      unfolding partial-equiv-def by blast
    from ‹partial-equiv A R› show trans R
      unfolding partial-equiv-def by blast
    show refl-on (R `` A) R
    proof (rule refl-onI)
      show R ⊆ R `` A × R `` A
      proof
        fix p
        assume p ∈ R
        obtain x y where p = (x, y) by fastforce
        moreover have x ∈ R `` A
          using ‹p ∈ R› ‹p = (x, y)› ‹partial-equiv A R›
            partial-equiv-def sym-def by fastforce
        moreover have y ∈ R `` A
          using ‹p ∈ R› ‹p = (x, y)› ‹R `` A ⊆ A› ‹x ∈ R `` A› by blast
        ultimately show p ∈ R `` A × R `` A by auto
      qed
    next
      fix y
      assume y ∈ R `` A
      from this obtain x where (x, y) ∈ R by auto
      from ‹(x, y) ∈ R› have (y, x) ∈ R
        using ‹sym R› by (meson symE)
      from ‹(x, y) ∈ R› ‹(y, x) ∈ R› show (y, y) ∈ R
        using ‹trans R› by (meson transE)
    qed
  qed
  ultimately show ∃ A'⊆A. equiv A' R by blast
next
  assume ∃ A'⊆A. equiv A' R
  from this obtain A' where A' ⊆ A and equiv A' R by blast
  show partial-equiv A R
  proof (rule partial-equivI)
    from ‹equiv A' R› ‹A' ⊆ A› show R ⊆ A × A
      using equiv-class-eq-iff by fastforce
    from ‹equiv A' R› show sym R
      using equivE by blast
    from ‹equiv A' R› show trans R
      using equivE by blast
  qed
qed
```

2.2 Construction of all Partial Equivalence Relations for a Given Set

```
definition all-partial-equivs-on :: 'a set ⇒ (('a × 'a) set) set
```

where
all-partial-equivs-on A =
do {
$k \leftarrow \{0..card\ A\}$;
$A' \leftarrow \{A'.\ A' \subseteq A \wedge card\ A' = k\}$;
$\{R.\ equiv\ A'\ R\}$
}

lemma *all-partial-equivs-on*:
assumes *finite* A
shows *all-partial-equivs-on* $A = \{R.\ partial\text{-}equiv\ A\ R\}$
proof
show *all-partial-equivs-on* $A \subseteq \{R.\ partial\text{-}equiv\ A\ R\}$
proof
fix R
assume $R \in$ *all-partial-equivs-on* A
from *this* **obtain** A' **where** $A' \subseteq A$ **and** *equiv* $A'\ R$
unfolding *all-partial-equivs-on-def* **by** *auto*
from *this* **have** *partial-equiv* $A\ R$
using *partial-equiv-iff* **by** *blast*
from *this* **show** $R \in \{R.\ partial\text{-}equiv\ A\ R\}$ **..**
qed
next
show $\{R.\ partial\text{-}equiv\ A\ R\} \subseteq$ *all-partial-equivs-on* A
proof
fix R
assume $R \in \{R.\ partial\text{-}equiv\ A\ R\}$
from *this* **obtain** A' **where** $A' \subseteq A$ **and** *equiv* $A'\ R$
using *partial-equiv-iff* **by** (*metis mem-Collect-eq*)
moreover have *card* $A' \in \{0..card\ A\}$
using ‹$A' \subseteq A$› ‹*finite* A› **by** (*simp add*: *card-mono*)
ultimately show $R \in$ *all-partial-equivs-on* A
unfolding *all-partial-equivs-on-def*
by (*auto simp del*: *atLeastAtMost-iff*)
qed
qed

2.3 Injectivity of the Set Construction

lemma *equiv-inject*:
assumes *equiv* $A\ R$ *equiv* $B\ R$
shows $A = B$
proof −
from *assms* **have** $R \subseteq A \times A$ $R \subseteq B \times B$ **by** (*simp add*: *equiv-type*)+
moreover from *assms* **have** $\forall x{\in}A.\ (x, x) \in R$ $\forall x{\in}B.\ (x, x) \in R$
by (*simp add*: *eq-equiv-class*)+
ultimately show *?thesis*
using *mem-Sigma-iff subset-antisym subset-eq* **by** *blast*
qed

lemma *injectivity*:
assumes $(A' \subseteq A \wedge card\ A' = k) \wedge (A'' \subseteq A \wedge card\ A'' = k')$
assumes $equiv\ A'\ R \wedge equiv\ A''\ R'$
assumes $R = R'$
shows $A' = A''$ $k = k'$
proof –
from ‹$R = R'$› *assms*(*2*) **show** $A' = A''$
using *equiv-inject* **by** *fast*
from *this assms*(*1*) **show** $k = k'$ **by** *simp*
qed

2.4 Cardinality Theorem of Partial Equivalence Relations

theorem *card-partial-equiv*:
assumes *finite A*
shows $card\ \{R.\ partial\text{-}equiv\ A\ R\} = Bell\ (card\ A + 1)$
proof –
let $?expr = do\ \{$
$k \leftarrow \{0..card\ A\};$
$A' \leftarrow \{A'.\ A' \subseteq A \wedge card\ A' = k\};$
$\{R.\ equiv\ A'\ R\}$
$\}$
have $card\ \{R.\ partial\text{-}equiv\ A\ R\} = card\ (all\text{-}partial\text{-}equivs\text{-}on\ A)$
using ‹*finite A*› **by** (*simp add*: *all-partial-equivs-on*)
also have $card\ (all\text{-}partial\text{-}equivs\text{-}on\ A) = card\ ?expr$
unfolding *all-partial-equivs-on-def* **..**
also have $card\ ?expr = (\sum k = 0..card\ A.\ (card\ A\ choose\ k) * Bell\ k)$
proof –
let $?S \ggg ?comp = ?expr$
{
fix *k*
assume k: $k \in \{..card\ A\}$
let $?expr = ?comp\ k$
let $?S \ggg ?comp = ?expr$
have *finite ?S* **using** ‹*finite A*› **by** *simp*
moreover {
fix A'
assume A': $A' \in \{A'.\ A' \subseteq A \wedge card\ A' = k\}$
from *this* **have** $A' \subseteq A$ **and** $card\ A' = k$ **by** *auto*
let $?expr = ?comp\ A'$
have *finite* A'
using ‹*finite A*› ‹$A' \subseteq A$› **by** (*simp add*: *finite-subset*)
have $card\ ?expr = Bell\ k$
using ‹*finite A*› ‹*finite* A'› ‹$A' \subseteq A$› ‹$card\ A' = k$›
by (*simp add*: *card-equiv-rel-eq-Bell*)
moreover have *finite ?expr*
using ‹*finite* A'› **by** (*simp add*: *finite-equiv*)
ultimately have $finite\ ?expr \wedge card\ ?expr = Bell\ k$ **by** *blast*

```
    } note inner = this
    moreover have disjoint-family-on ?comp ?S
      by (injectivity-solver rule: injectivity(1))
    moreover have card ?S = card A choose k
      using ‹finite A› by (simp add: n-subsets)
    ultimately have card ?expr = (card A choose k) * Bell k (is - = ?formula)
      by (subst card-bind-constant) auto
    moreover have finite ?expr
      using ‹finite ?S› inner by (auto intro!: finite-bind)
    ultimately have finite ?expr ∧ card ?expr = ?formula by blast
  }
  moreover have finite ?S by simp
  moreover have disjoint-family-on ?comp ?S
    by (injectivity-solver rule: injectivity(2))
  ultimately show card ?expr = (∑ k = 0..card A. (card A choose k) * Bell k)
    by (subst card-bind) auto
 qed
 also have ... = (∑ k≤card A. (card A choose k) * Bell k)
   by (auto intro: sum.cong)
 also have ... = Bell (card A + 1)
   using Bell-recursive-eq by simp
 finally show ?thesis .
qed

end
```

References

[1] L. Bulwahn. Cardinality of set partitions. *Archive of Formal Proofs*, Dec. 2015. http://www.isa-afp.org/entries/Card_Partitions.shtml, Formal proof development.

[2] L. Bulwahn. Spivey's generalized recurrence for bell numbers. *Archive of Formal Proofs*, May 2016. http://www.isa-afp.org/entries/Bell_Numbers_Spivey.shtml, Formal proof development.

[3] N. J. A. Sloane. A000110: Bell or exponential numbers: number of ways to partition a set of n labeled elements. In *The On-Line Encyclopedia of Integer Sequences*. https://oeis.org/A000110.

[4] Wikipedia. Equivalence relation — wikipedia, the free encyclopedia, 2016. [Online; accessed 23-May-2016].